A Computationally Efficient Triple Matrix Product for a Class of Sparse Schur-complement Matrices

Two Implementations: One-Phase vs. Two-Phase Schemes

We compare two approaches to compute the triple-product. While one-phase scheme has an advantage over two-phase scheme by using a knowledge on the structure of matrix, the summation of sparse matrices becomes bottleneck.

Hence, we propose a row-based one-phase scheme, where the summation of sparse matrices is replaced by the summation of sparse vectors, which can be computed efficiently using a sparse accumulator.

In two-phase scheme, where the summation of sparse matrices becomes bottleneck.

We also improved the performance of the row-based one-phase scheme through use of additional data structures.

Two-Phase scheme

\[ P = \text{mult}(A, \text{mult}(L, H)) \]

In Computing \( C = \text{mult}(A, B) \)

For \( B_i \) each column of \( B \)

For each nonzero of \( B_{ij} \) do the following

- Counting the number of nonzeros in \( B \) and \( P \)
- Computing the structure of matrix \( B \)
- Constructing row-major structure of \( A \) and \( B \)
- Generating \( A' \)
- Counting the number of nonzeros in \( P \) and \( Q \)

One-Phase Scheme

This scheme can take advantage of known structure of \( H \), and symmetry of \( P \) using the following equation.

\[ P = \sum_{i} (A_i H_i') + \sum_{i,j} (B_i H_i' B_{j} H_j') \]

Drawback: a summation of sparse matrices is slow.

Efficient Sparse Vector Addition using a sparse accumulator

A sparse accumulator is used in one-phase and row-based two-phase schemes.

Preprocessing

- In one-phase scheme
  - counting the number of nonzeros in \( B \) and \( P \) (to determine the amount of memory allocation)
  - computing the structure of matrix \( B \)
  - constructing row-major structure of \( A \) and \( B \)
- In two-phase scheme
  - generating \( A' \)
  - counting the number of nonzeros in \( P \) and \( Q \)

Performance

We predict lower and upper bounds of the execution time for one-phase and two-phase scheme using our memory model, and it is confirmed by measurements that one-phase scheme has advantage of execution time and memory over two-phase scheme.

In addition, the preprocessing cost is lower in one-phase scheme.

Memory Performance Modeling

- Memory Access
  - Dominant factor in One-phase scheme: access of elements of \( A \) in \( A \times P \):
    \[ \sum_{i} \sum_{j} (A_i A_j B_{ij}) \]
  - Dominant factor in Two-phase scheme: access of elements of \( A \) in \( A \times B \):
    \[ \sum_{i} \sum_{j} (A_i A_j B_{ij}) \]
- Cache Miss
  - For sequentially accessed elements, spatial locality is assumed to be exploited.
- Execution Time
  - \( T = \text{access} \times \text{memory access} \times \text{memory} \times \text{level} - \text{cache} \)
  - \( a \) latency of level - cache
  - \( a \) latency of memory
  - \( k \) level of cache
  - \( M \) cache miss in level - cache

Efficient Sparse Vector Addition

Utilizing the Symmetry of \( P \)

\[ P = \sum_{i,j} a_{ij} A_{ij}' + \sum_{i,j} b_{ij} B_{ij}' \]

- In computing \( a_{ij} A_{ij}' \), compute \( a_{ij} A_{ij}' \)
- by keeping an array of indices pointing to each \( A_{ij} \)'s next nonzero element, unnecessary access to \( A_{ij} \)'s is avoided.

Example Matrix Set

from Circuit Design Application

<table>
<thead>
<tr>
<th>set</th>
<th>( n(A) )</th>
<th>( n(Q) )</th>
<th>( n(H) )</th>
<th>( # fop. )</th>
<th>Mem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8648</td>
<td>42750</td>
<td>361K</td>
<td>150K</td>
<td>11M</td>
</tr>
<tr>
<td>2</td>
<td>14782</td>
<td>77446</td>
<td>361K</td>
<td>150K</td>
<td>11M</td>
</tr>
<tr>
<td>3</td>
<td>612150</td>
<td>977K</td>
<td>528K</td>
<td>31M</td>
<td>29M</td>
</tr>
<tr>
<td>4</td>
<td>39768</td>
<td>217020</td>
<td>1913K</td>
<td>1028K</td>
<td>50M</td>
</tr>
<tr>
<td>5</td>
<td>41392</td>
<td>244501</td>
<td>1633K</td>
<td>963K</td>
<td>50M</td>
</tr>
</tbody>
</table>

Measured Performance

Speedup

Achieved Mflop rate

Achieved Mflop rate relative to execution time in two-phase scheme

Conclusion

- Performance tuning of higher level sparse matrix operation than matrix-vector multiplication
- Speedup up to 2.1x
- Less than half memory requirement
- An example of algebraic transformation is used for performance tuning
- Knowledge on the special structure of the matrix is used for the algebraic transformation.